ECE 443/518 – Computer Cyber Security
Lecture 28 Malware

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

November 24, 2025

Outline

Malware

Stack Overflow

Blue Pill

Reading Assignment

- This lecture: ICS 19
- Next lecture: Hardware Security

Outline

Malware

Stack Overflow

Blue Pill

Malware

- A piece of software running in a computer system that may impact normal operation or cause damage.
  - Virus, worm, adware, ransomware, etc.
- From the viewpoint of secure policy, the system is actually in an insecure state.
  - Computer systems are so complicated nowadays that human being cannot really decide if a system state is secure or not.
- To make matters worse, malware may exploit errors and bugs in OS, applications, and services to bypass access control.

Life Cycle

1. Dormant phase
   - Infect the host system.
   - Survive reboot, removal, or even reinstallation.
2. Propagation phase
   - Infect other systems via a shared media.
3. Triggering phase
   - Being activated by various system or network events.
   - Lay low to avoid detection.
4. Execution phase
   - Perform predefined malicious behavior.

- Defense mechanisms can be designed to address various portion of the life cycle.
- Attackers adapt to computing trend when creating malware.
  - Skip and combine phases.

Dormant Phase

- Infection: bypass existing access control mechanism.
  - Exploit human ignorance or error via social engineering.
  - Inject code via bugs.
- Survival: persistence of (binary) program
  - Files: hidden files, fake system files, end of other files, etc.
  - File systems: unused partition area, NTFS data streams, etc.
  - Firmware: disk controller, network controller, etc.
- Defense
  - Educate non-technical people.
  - Improve system usability to reduce human error.
  - Improve software quality to reduce bugs.
  - Scan storage area for suspicious data.

Propagation Phase

- Correspond to infection during dormant phase.
- Propagation consumes system and network resources and may be detected by monitoring so.
  - The most hostile malware will attempt to propagate silently as much as possible without being caught.
  - In practice, many malwares won't care since they target at common people.
- Defense: isolation and containment
  - So infected or suspicious systems won't cause harm to others.
  - Via firewall, virtual machine, etc.

Triggering Phase

- Persistence of the malware program ties to how it is triggered.
- Malware programs may also be triggered via remote control.
  - E.g. coordinated for a DDoS attack.
- The malware would need to consume resource to detect triggering events while laying low to avoid detection.
  - Short burst: awake for a very short time everytime something happens, e.g. as a browser plugin that runs everytime a page loads.
  - Long live: monitor system status continuously, e.g. as a fake system process or reside in a valid system process.
- Defense: more places to scan for suspicious data.
  - A scan could trigger the malware to stop the scan itself.
  - Isolation and containment help if the malware is triggered remotely.

Execution Phase

- Damages
  - Physical damage to components and attached devices, e.g. CIH, Stuxnet.
  - Exhaust resources, e.g. Morris worm.
  - Data erasure, e.g. ransomware.
  - Leakage of sensitive data or access right.
- Defense
  - Fail-safe mechanisms for components and attached devices.
  - Backup data.
  - Isolation and containment.

Layered Defense

- Multiple defense mechanisms are required to be integrated in a layered fashion for a complex computer system.
  - Users with different knowledge of technology and sense of security.
  - Systems running programs with different trust levels.
- Isolation and containment within a system.
  - So an infected subsystem will have less chance impact others.
  - Fine grained access control: programs are only granted permissions to a minimal number of subsystems.
- Communication only via predefined interfaces.
  - Explicit flow of information.
  - Allow more focused efforts to locate bugs.

Typical Malware: Trojan Horses

- Trojan Horses: a program with an overt (documented or known) effect and a covert (undocumented or unexpected) effect.
- Life cycle
  - Trojan houses usually involve a lot of techniques for survival and triggering.
  - Trojan horses open doors for other programs to propagate and execute.

Typical Malware: Computer Viruses

- Virus: the malware cannot exist by itself and must attach to existing programs.
- Infection: computer virus modifies other executable files
  - Append itself to the end.
  - Call it when the program starts.
- Infection simplifies dormant and triggering phases.
  - But this makes scanning of suspicious data easier via use of patterns and hashes.
- Propagation is usually achieved by infecting executables on removable medias.
  - Most OS are aware of such risk and usually will notify users so.

Typical Malware: Computer Worms

- Worms propagate.
    - Via network.
- How worms propagate?
    - Buggy network service programs, in particular old unpatched services with known vulnerabilities.
    - Buggy browsers and careless email users.
- Worms by themselves usually do no harm during execution other than exhausting resources.
    - But attackers usually combine trojan horses with worms to create malwares strong at dormant, propagation, and triggering – opening doors to execute any malicious code.

Outline

Malware

Stack Overflow

Blue Pill

Stack Overflow

- A typical software bug.
- A service program runs at a higher trust level that allows clients to access resources.
    - Locally or remotely, both via predefined communication channels and protocols for access control.
- Attacks as clients exploit bugs in the service program to inject codes that run within it at the higher trust level.
    - Effectively bypass access control mechanisms.

The Call Stack

- Portion of the memory is managed as a stack to facilitate function calls.
    - Stack: last in, first out.
- The stack entries are organized logically as stack frames.
    - Each stack frame contains information regarding a particular function call.
    - Arguments, return address, local variables.
    - Details depend on the calling convention.

Stack Frame Example

```
void test(int a) {
    int v[10];
    printf("%d\n", v[a]);
}
void caller() {
    test(20);
    printf("Done.\n");
}
```

- Assume all arguments are passed via the stack.
- When `caller` calls `test`
  - Push 20 for the argument `a`.
  - Push return address of test, i.e. address of `printf("Done.\n");`.
  - Jump to first instruction of `test`.
- Inside `test`
  - Push 10 `int` for the local variable `v`.
  - Visit stack to obtain `a`.
  - Visit and print `v[a]`.
  - Pop 10 `int` to destroy the local variable `v`.
  - Pop the return address and jump to it.

The Vulnerability

```
void test(int a) {
    int v[10];
    printf("%d\n", v[a]);
}
void caller() {
    test(20);
    printf("Done.\n");
}
```

- But `v` only contains 10 elements, what does `v[a]` refer to for $a < 0$ or $a \geq 10$?
  - The C language doesn't check for that for performance reasons.
  - Depend on where stack grows.
- If stack grows down, e.g. on x86 processors.
  - `v[a]` for $a < 0$ refers to a lower memory location, which contains garbage not belonging to the stack.
  - `v[a]` for $a \geq 10$ refers to a higher memory location that may be used by a stack entry.

The Attack

```
void test(int a, int b) {
    int v[10];
    v[a] = b;
}
void caller() {
    int a, b;
    scanf("%d %d", &a, &b);
    test(a, b);
    printf("Done.\n");
}
```

- Attacks inject code by overwriting stack entries if the implementation does not check input values.
- Those modified stack entries will be interpreted by other portion of the program to trigger the actual code execution.

Shellcode Exploit

- Modify portion of the stack to include a shellcode, e.g. a small program.
- Modify the return address to point to the shellcode.
- When the function returns, the shellcode runs within the original program.
    - It could simply attack from there, or open doors for further attacks, e.g. by creating an account if the original program runs as root.
- Defense: executable space protection
    - Mark the memory portion used by stack as non-executable.
    - Supported by processor, enforced By OS

Return-to-libc Attack

- Modify the return address to point to an existing function.
  - E.g. one from libc where almost all programs make use of.
- Need to modify additional entries on the stack so that the libc function is called with meaningful arguments.
  - E.g. modify the return address and additional stack entries to call `system("/bin/sh")` so that the attacker can access the shell program.
- Defense: address space layout randomization (ASLR)
  - Load functions to different addresses everytime so attackers won't know their addresses.
  - Enforced when OS loads a program to execute.

Other Similar Attacks

- Heap overflow attacks: modify stack indirectly by exploiting buggy accesses to memory blocks allocated dynamically.
- Integer overflow attacks: access portion of memory through a pointer not for such purpose.
- Many of such problems also cause software to fail in general – we may borrow from software practices to reduce their chances to happen.
  - Use a safer language that checks indices for array visits and that managers dynamic allocations automatically – actually almost all languages designed in the past 30 years are doing so.
  - Test a lot.

Outline

Malware

Stack Overflow

Blue Pill

Hypervisor and Virtual Machine

- For security purpose, we generally depend on virtual machines to provide necessary isolation and containment.
  - Modern OS does provide certain level of isolation between kernel and applications.
  - But they are mostly for accidental errors but not security risks.
- Virtual machines are supported by both hardware and a special piece of software called hypervisor.
  - Similar to OS, hypervisors make it possible to virtualize/share hardware resources.
  - Hypervisors are not as complicated as OS so that there will be less chance to have buggy services.
- Since hypervisor controls the hardware, it knows anything running on the processor.
  - What if hypervisors are compromised?
  - What if processors/hardware are compromised?

Blue Pill

- A conceptual hypervisor-based malware.
- The malware itself is the hypervisor.
  - Small in size, but may perform any operation on the OS running on top of it.
- Could the OS running under full control of a hypervisor know it is running under a hypervisor?

Summary

- Malware works within access control but may bypass it via errors and bugs.
- More powerful malwares may be created by combining malwares strong for different life cycle phases, requiring defense mechanisms to be more effective.
- However, it is human beings that control the computer system so one should not underestimate the human risk factor even a very strong defense mechanism is used.